Lock (computer science)

From Wikipedia, the free encyclopedia
Revision as of 22:03, 12 November 2003 by Zoicon5 (Talk | contribs)
(diff) ← Older revision | Current revision | Newer revision → (diff)

In computer science, a **lock** is a mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution. Locks are one way of enforcing concurrency control policies.

Generally, locks are **advisory locks**, where each thread cooperates by acquiring the lock before accessing the corresponding data. Some systems also implement **mandatory locks**, where attempting unauthorized access to a locked resource will force an exception in the entity attempting to make the access.

A semaphore is the simplest type of lock. No distinction is made between shared (read only) or exclusive (read and write) modes. Other schemes provide for a shared mode, where several threads can acquire a shared lock for read-only access to the data. Other modes such as shared, exclusive, intend-to-exclude and intend-to-upgrade are also widely implemented.

Independent of the type of lock chosen above, locks can be classified by what happens when the lock strategy prevents progress of a thread. Most locking designs block the execution of the process requesting the lock until it is allowed to access the locked resource. A spinlock is a lock where the thread simply waits ("spins") until the lock becomes available. It is very efficient if threads are only likely be blocked for a short period of time, as it avoids the overhead of operating system process re-scheduling. It is wasteful if the lock is held for a long period of time.

Locks require hardware support for safe implementation. This is usually provided by an "atomic" "test-and-set" or "compare-and-swap" instruction. Some uniprocessor architectures have in the past used uninterruptable sequences of instructions, using special instructions or instruction prefixes. However, this does not scale to multiprocessor environments. Proper support for locks in a multiprocessor environment can require quite complex hardware and/or software support, with substantial synchronization issues.

Careless use of locks can result in deadlock, where multiple processes are unable to proceed. A number of strategies can be used to avoid deadlock, both at design time and at run-time.

See also:

- ACID properties
- semaphore
- latch
- monitor
- mutex
- critical section